Principal Component Analysis

Gerardo De la O

‘2022

Refining Models with PCA

In this section, we will use Principal Component Analysis to refine our previous linear model for crime. To remain consistent and to draw comparisons, we include the necessary code used to generate the previous model along with a very basic summary of steps.

Preparation

The purpose of this step is to understand the data and perform some basic cleanup. Removing outliers and scaling the data are still applicable when using PCA (scaling is actually necessary), so we retain the following steps:

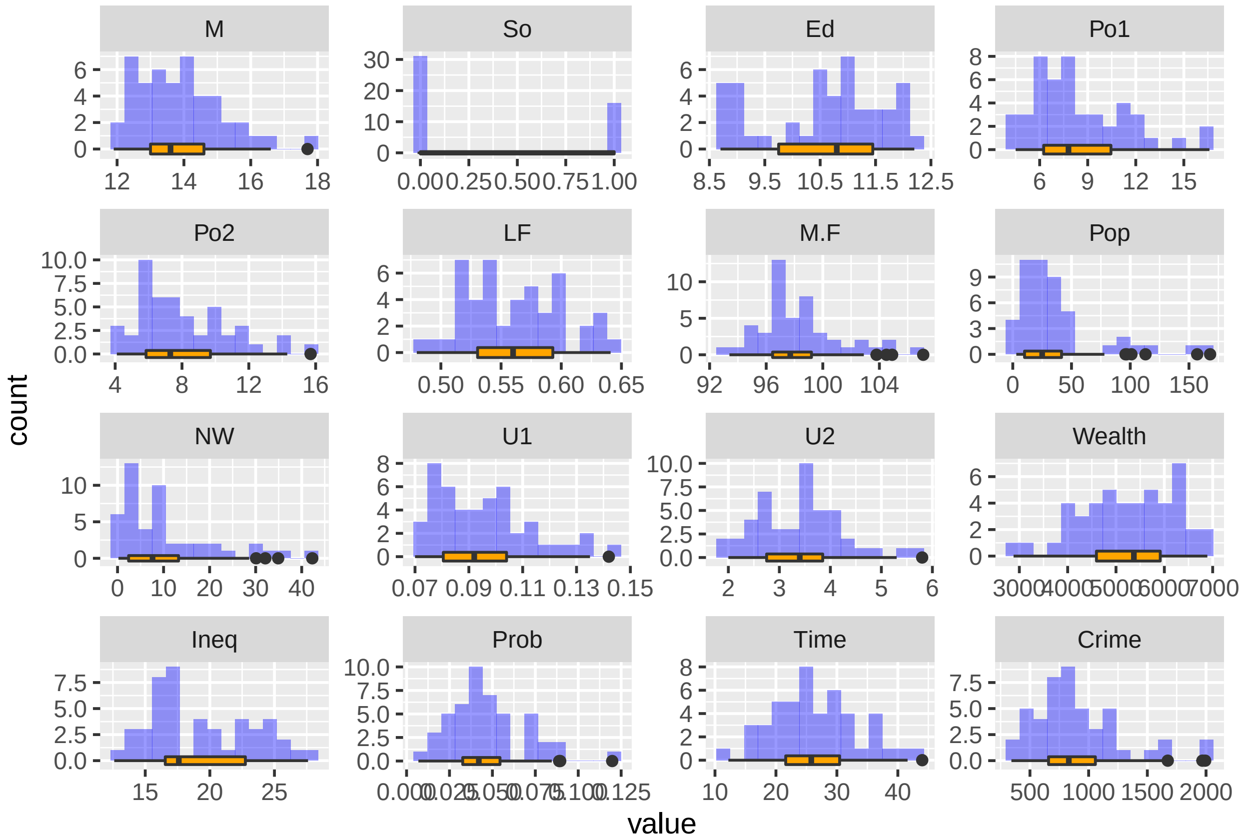
1. Remove rows where a z-score is greater than 3 (outside the 99.7th percentile). In this case, we favor a model that behaves well within its bounds over something super general.(5 total points removed)
2. Standardize the data to $\mu = 0$ and sd $= 1$

```
library(tidyverse)
library(reshape2)
library(caret)

#import data
filename = 'http://www.statsci.org/data/general/uscrime.txt'
data <- read.csv(filename, sep = '\t' ) #tab delimited

# Melt to plot easily
melted_data <- melt(data)

ggplot(data=melted_data, aes(x = value)) +
    geom_histogram(bins=15, fill='blue', alpha=0.4) +
    geom_boxplot(fill='orange') +
    facet_wrap(~variable, scales = "free")
```




```r
# Outlier removal
z_scores <- as.data.frame(sapply(data, function(data) (abs(data-mean(data))/sd(data))))
delete = rowSums(z_scores > 3)

df <- data[!delete, ]
print(paste('Removed rows: ', dim(data)[1]-dim(df)[1]))
```

```
## [1] "Removed rows:  5"
```

```r
# Scale the data (only the predictors)
stdize = preProcess(df[,-16], method = c("center", "scale"))
df_sc = round(predict(stdize, df),4)

# Build and Scale the example input vector
testVal = df_sc[FALSE,-16]
testVal[nrow(testVal) + 1,] = c(14,0,10,12,15.5,.640,94,150,1.1,.120,3.6,3200,20.1,.04,39)
testVal_sc = predict(stdize, testVal)
```

Models

Original Model In the previous assignment, we ran a correlation analysis to illustrate the concept of multicollinearity and found that quite a few of the variables are highly correlated. This process is not needed when using PCA, since the PCA inherently removes correlation. In that analysis,however, we ended up removing Po2. Note that we use the same seed as the one used in the last HW to ensure the model and predictions are exactly the same.

```r
library (broom)
library (Metrics)

set.seed(1)
# input = df_sc
input = subset(df_sc, select=-c(Po2))

# generate train/test sets
smp_size <- floor(.80 * nrow(input))
sampler <- sample(seq_len(nrow(input)), size = smp_size)
train <- input[sampler, ]
test <- input[-sampler, ]

# train  model
model <- lm(Crime ~ .,data=train)
fit = model$fitted.values

# Fit model to test set
test_pred = predict(model, test)

# Compute RMSE on test set
RMSE = rmse(actual = df_sc$Crime, predicted = test_pred)
print(paste('RMS Error:', RMSE))
```

```
## [1] "RMS Error: 486.263437697816"
```

```r
# Model Details
output <- tidy(model)
print(summary(model))
```

```
##
## Call:
## lm(formula = Crime ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -229.70  -71.06  -17.30   87.85  333.04
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   914.11      31.62  28.909  < 2e-16 ***
## M             102.27      53.57   1.909  0.07235 .
## So            -39.69      76.76  -0.517  0.61141
## Ed            285.03      76.95   3.704  0.00162 **
## Po1           256.16      69.39   3.692  0.00167 **
## LF            -92.56      87.07  -1.063  0.30177
## M.F            93.10      74.31   1.253  0.22626
## Pop            32.42      51.58   0.628  0.53758
## NW             72.16      58.54   1.233  0.23354
## U1           -161.31      90.52  -1.782  0.09160 .
## U2            176.98      79.27   2.233  0.03851 *
## Wealth        -62.47      95.28  -0.656  0.52033
## Ineq          193.08      92.65   2.084  0.05169 .
## Prob          -97.08      68.37  -1.420  0.17272
```

```
## Time           20.58      59.99   0.343  0.73550
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 173.8 on 18 degrees of freedom
## Multiple R-squared:  0.8927, Adjusted R-squared:  0.8092
## F-statistic: 10.69 on 14 and 18 DF,  p-value: 5.148e-06
```

```
# Test Input Prediction
TI = predict(model, newdata=testVal_sc)
print(paste('Given Input Prediction: ', TI))
```

```
## [1] "Given Input Prediction:  1026.89779880628"
```

Regression with PCA

Here, we apply the PCA process with the goal of eliminating correlation as well as to simplify our model a bit.

PCA Transform The first step is to calculate our V transform for our X matrix. After applying PCA, we want a data frame ready to be plugged into the regression function.

There are 3 things to note for this step:

1. We do NOT transform the Y response. This saves us the need to un-scale the data later. Also, PCA has nothing to do with the response, so this does not affect the process.(Contrast this with a method like SVM, where the response does matter)
2. From the summary, we can see the "Cumulative Proportion" (total variance) explained after adding each principal component. We choose n=8 for this exercise because that feature set accounts for 95% of the variance. Essentially, we are halving the number of predictors for only a 5% loss in variance explained! Some discretion is applicable here, 5 or 10 would have worked just as well.
3. We do NOT scale the data in the call to prcomp() because we already scaled it in the previous step. If we add, scale=TRUE, the result would be the same but it would be a redundant computation.

```
# Run PCA (Exclude the response Y)
pca = prcomp(df_sc[,-16])
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5    PC6     PC7
## Standard deviation     2.4531 1.6757 1.4086 1.08162 0.92919 0.8660 0.59572
## Proportion of Variance 0.4012 0.1872 0.1323 0.07799 0.05756 0.0500 0.02366
## Cumulative Proportion  0.4012 0.5884 0.7207 0.79865 0.85621 0.9062 0.92987
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.52136 0.49079 0.42809 0.38999 0.28544 0.25585 0.22500
## Proportion of Variance 0.01812 0.01606 0.01222 0.01014 0.00543 0.00436 0.00337
## Cumulative Proportion  0.94799 0.96405 0.97627 0.98641 0.99184 0.99620 0.99958
##                           PC15
## Standard deviation     0.07966
## Proportion of Variance 0.00042
## Cumulative Proportion  1.00000
```

```
# Linear Transform "V"
pca_transform = pca$rotation
```

```
# Apply Transform to X (Keep n principal components)
n = 8
trans = as.matrix(pca_transform[,1:n])
df_pca = as.data.frame(as.matrix(df_sc[,-16]) %*% trans)

# Add back the Y vector, then its ready for modeling!
df_pca['Crime'] = df_sc[,16]

head(df_pca,4)
```

```
##         PC1        PC2         PC3        PC4        PC5        PC6        PC7
## 1 -4.836327 -0.7416120 -0.80850253 -0.8242217  0.3908102 -0.3278081 -0.5520205
## 2  1.337527  0.3005018 -0.09683075 -0.9118781 -1.1447066  0.2412801  0.1324056
## 3 -4.682142  0.7030268 -0.09603291 -0.1315631  0.5811696 -0.6103584  0.4501127
## 5  2.090813  1.0216723  1.23104838 -0.9144586  0.2198949  0.3858544 -0.3209106
##          PC8 Crime
## 1 -0.3124147   791
## 2  0.2884191  1635
## 3 -0.1701193   578
## 5  0.2711793  1234
```

PCA Model Now that we have our nice, transformed data frame, we can model the results. The exact same logic applies to this step as it did with the un-factored input.

There are 3 things to note for this step:

1. The RMSE error against the test set for the PCA model is lower than our previous model! This means that PCA worked in making our model better.
2. It looks like PCA 6,7 and 8 are not statistically significant, opening the option to use even less PCs.
3. The predictors are still transformed, so at this point they are not easily interpreted. We will get to that later.

```
set.seed(1)
# input = df_pca
input = df_pca

# generate train/test sets
smp_size <- floor(.80 * nrow(input))
sampler <- sample(seq_len(nrow(input)), size = smp_size)
train <- input[sampler, ]
test <- input[-sampler, ]

# train  model
model_pca <- lm(Crime ~ .,data=train)
fit = model_pca$fitted.values

# Fit model to test set
test_pred = predict(model_pca, test)

# Compute RMSE on test set
RMSE = rmse(actual = df_sc$Crime, predicted = test_pred)
print(paste('RMS Error:', RMSE))
```

```
## [1] "RMS Error: 472.370319064879"
```

```r
# Model Details
output <- tidy(model_pca)
print(summary(model_pca))
```

```
##
## Call:
## lm(formula = Crime ~ ., data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -363.3 -130.4    2.5  109.3  400.1
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   909.42      38.21  23.799  < 2e-16 ***
## PC1            47.16      15.12   3.119 0.004667 **
## PC2           -96.37      24.00  -4.015 0.000508 ***
## PC3            22.92      25.85   0.887 0.383987
## PC4          -196.60      37.23  -5.280 2.05e-05 ***
## PC5          -174.45      41.32  -4.222 0.000301 ***
## PC6           -25.98      43.22  -0.601 0.553337
## PC7           -30.05      71.56  -0.420 0.678251
## PC8           -31.71      70.64  -0.449 0.657506
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 210.9 on 24 degrees of freedom
## Multiple R-squared:  0.7892, Adjusted R-squared:  0.719
## F-statistic: 11.23 on 8 and 24 DF,  p-value: 1.827e-06
```

PCA Interpretation Now that we have our model, we need to make it interpretable and we need to make it able to take in new data in the original vector space! If we had to apply a PCA transformation to every new data point that went into the model, we (or the server) would have a bad time.

2 final notes for this step which are answers to the HW prompt:

1. Matrix-Matrix operations make going back and forth relatively easy. The tranformed PCA coefficients are shown below. Note that all but 2 of the predictors are positively associated with crime!
2. The prediction for the given input vector is 1340.1, considerably higher than our initial value of 1026.9. This prediction would put crime for this element on the higher end of the spectrum, but still within "normal" range.

```r
# Transform coefficients to original space

# This makes our Coefficients from PCA interpretable!
pca_coeffs = as.matrix(as.data.frame(model_pca$coefficients)[-1,])

real_coeffs = trans %*% pca_coeffs







real_coeffs
```

```
##                [,1]
## M       89.663949
## So      73.704224
## Ed      32.301894
## Po1    124.576866
## Po2    116.298589
## LF      39.049032
## M.F    114.224685
## Pop     25.496226
## NW     126.745188
## U1      -8.565143
## U2      22.246306
## Wealth  45.195904
## Ineq    20.464800
## Prob   -51.518181
## Time    56.340021
```

```r
# Test Input Prediction
testInput = as.matrix(testVal_sc)
b = as.numeric(model_pca$coefficients['(Intercept)'])

# Ax + b
pred = t(real_coeffs) %*% t(testInput) + b
print(paste('Prediction:', pred))
```

```
## [1] "Prediction: 1340.99772450062"
```